Predicting the Measured Power

October 1, 2022

1 Using Linear Regression to determine TxPowerLevel of AirTag

The Transmission Power Level (txPowerLevel) is a constant and refers to the signal strength received by a receiver from a broadcasting device when it is 1 meter away from the transmitting device (estimote.com, 2022). It is also referred to as Measured Power. It can be used to estimate the distance away from the transmitting device with the help of signal strength (RSSI) (Shah, 2022). It is sometimes set and determined by the manufacturers. Here, since it has not been set in the AirTag, I am using a linear regression model to estimate the signal strength at 1m.

1.1 Importing the Data

The experiment data was imported. This data is made up of the average dBm signal strength measured at various lengths of 0.02m, 0.10m, 0.25m, 0.50m, 0.75m, 1.00m, 1.50m, 2.00m, 2.50m, 3.00m, 5.00m, 7.00m and 10.00m

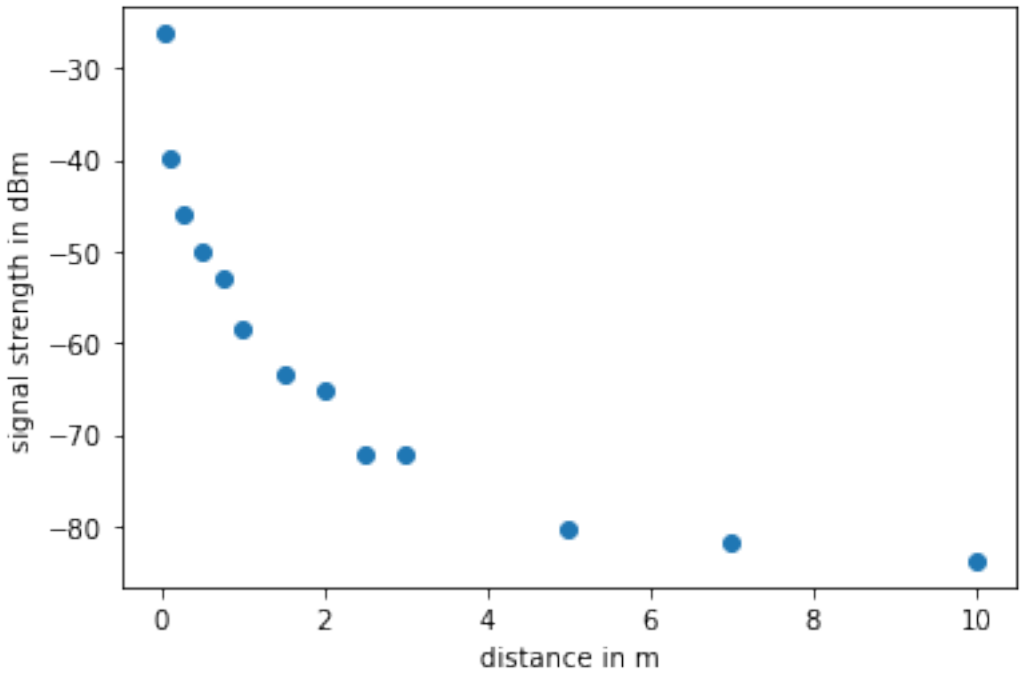
[3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

signal_strength = pd.read_csv("D:\Desktop\MSC\DISSERTATION\signal strength␣
↪experiment.csv")

signal_strength
```

[3]:

	distance	rssi
0	0.02	-26.25
1	0.10	-39.75
2	0.25	-46.00
3	0.50	-50.00
4	0.75	-53.00
5	1.00	-58.50
6	1.50	-63.50
7	2.00	-65.25
8	2.50	-72.00
9	3.00	-72.25
10	5.00	-80.25
11	7.00	-81.75
12	10.00	-83.75

1.2 Plotting a scatter plot of the data

[4]:
```
plt.scatter(signal_strength['distance'],signal_strength['rssi'])
plt.ylabel("signal strength in dBm")
plt.xlabel("distance in m")
plt.show()
```



1.3 Creating a linear regression model to fit the data

The independent variable is the "distance" attribute and the dependent variable is the "rssi" attribute, which we want to predict(estimate) for 1m

[5]:
```
#code inspired from statology.org

from statsmodels.formula.api import ols

model = ols('rssi ~ distance', data=signal_strength).fit()

print(model.params)
```

```
Intercept   -48.674825
distance     -4.743524
dtype: float64
```

[6]:
```
#code inspired from Principles of Data Science Lab Feedbacks

from sklearn.linear_model import LinearRegression
model1 = LinearRegression()

model1.fit(signal_strength[['distance']],signal_strength['rssi'])
print('intercept:', model1.intercept_)
print('slope:', model1.coef_)
```

```
intercept: -48.67482470990923
slope: [-4.74352406]
```

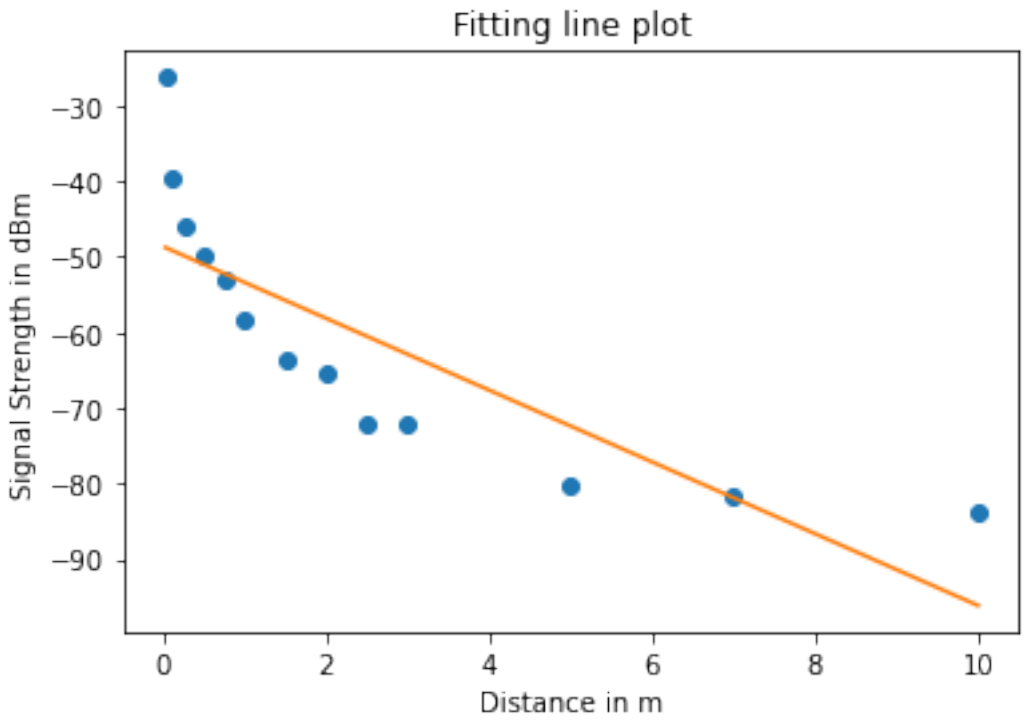
1.4 Validation of Model

This is to validate whether the model fits the data and is optimum to use for prediction

1.4.1 Line of best fit

This is the first validation analysis to observe whether the lineaar regression model fits the data

[7]:
```
#code inspired from kite.com

plt.plot(signal_strength['distance'], signal_strength['rssi'], 'o')
plt.plot(signal_strength['distance'], model1.coef_*signal_strength['distance']␣
 ↪+ model1.intercept_)
plt.title("Fitting line plot")
plt.xlabel("Distance in m")
plt.ylabel("Signal Strength in dBm")
plt.show()
```



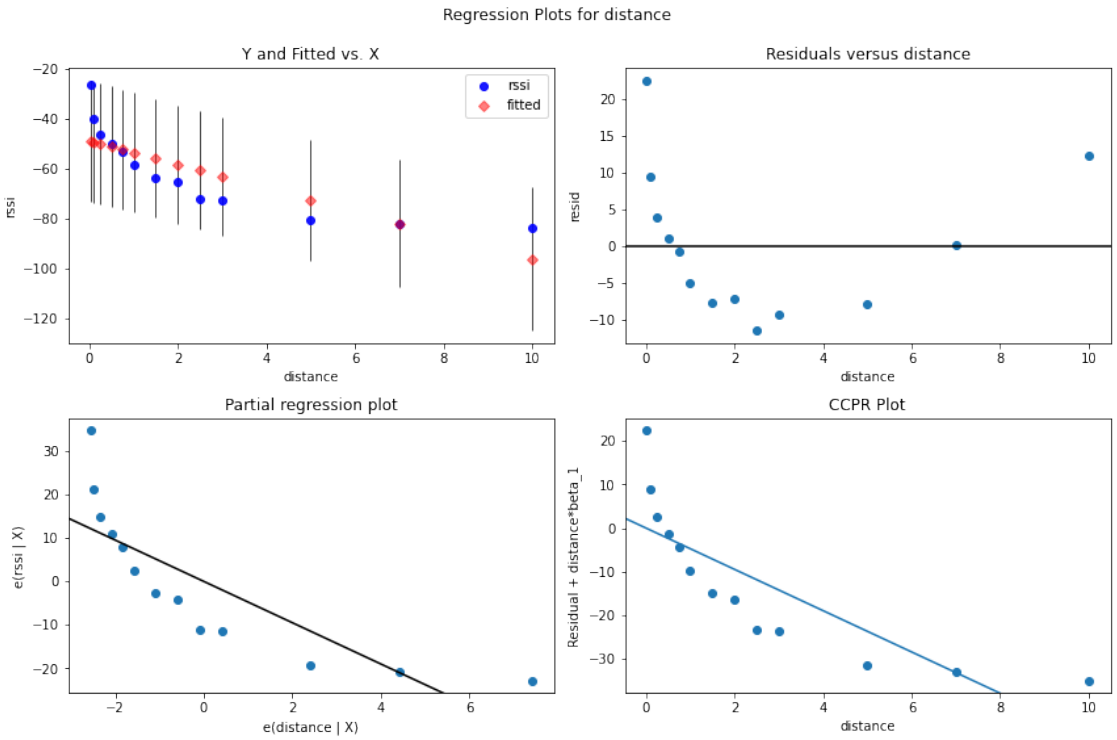
From initial observation, it looks to fit the data.

1.4.2 Plotting Residual Plot

We also need to plot a residual plot to assess whether or not the data fits by checking for heteroscedasticity of residuals.

"In regression analysis, heteroscedasticity refers to the unequal scatter of residuals or error terms." (Statology, 2022).

[8]:
```
#code inspired from statology.org

import statsmodels.api as sm

fig = plt.figure(figsize=(12,8))

#produce regression plots
fig = sm.graphics.plot_regress_exog(model, 'distance', fig=fig)
```



The top right plot displays the residual plot and since the residuals are randomly scattered around zero, this is an indication that heteroscedasticity is not a problem with the independent variable, distance, and that the data fits the model

1.4.3 Finding Coefficient of Determination (²)

The coefficient of determination tells which amount of variation in "rssi" can be explained by the dependence on "distance" using the linear regression model constructed.

[9]:
```
#code inspired from kite.com

import numpy as np

correlation_matrix = np.corrcoef(signal_strength['distance'],␣
 ↪signal_strength['rssi'])
correlation_xy = correlation_matrix[0,1]
r_squared = correlation_xy**2

print(r_squared)
```

```
0.6778668358892557
```

Larger ² indicates a better fit and means that the model can better explain the variation of the output with different inputs. The ² value shows an optimum fit, ie there is no overfitting and underfitting

1.5 Prediction of the TxPowerLevel

Now to predict the signal strength at 1m, which represents the txPowerLevel

[10]:
```
#code inspired from realpython.org

predict_1m = np.array([1, 5, 10,20]).reshape((-1, 1))

prediction = model1.predict(predict_1m)
print('predicted response:', prediction, sep='\n')
```

```
predicted response:
[ -53.41834877  -72.39244499  -96.11006527 -143.54530583]
```

As can be seen from the prediction, the estimated Measured Power is -53.418 dBm

Measured Powerl: -53.418 dBm

1.5.1 References

Estimote.com. 2022. [online] Available at: https://community.estimote.com/hc/en-us/articles/201636913-What-are-Broadcasting-Power-RSSI-and-other-characteristics-of-a-beacon-s-s

Statology. 2022. Understanding Heteroscedasticity in Regression Analysis - Statology. [online] Available at: https://www.statology.org/heteroscedasticity-regression/

Shah, R., 2022. Convert RSSI Value of the BLE (Bluetooth Low Energy) Beacons to Meters. [online] Medium. Available at: https://medium.com/beingcoders/convert-rssi-value-of-the-ble-bluetooth-low-energy-beacons-to-meters-63259f307283

[ ]: